N91-20681

EXODUS: Integrating Intelligent Systems for Launch Operations Support

Richard M. Adler and Bruce H. Cottman
Symbiotics, Inc.
875 Main Street
Cambridge, MA 02139

Abstract

NASA Kennedy Space Center (KSC) is developing knowledge–based systems to automate critical operations functions for the Space Shuttle fleet. Intelligent systems will monitor vehicle and ground support subsystems for anomalies, assist in isolating and managing faults, and plan and schedule Shuttle Operations activities. These applications are being developed independently of one another, using different representation schemes, reasoning and control models, and hardware platforms. KSC has recently initiated the EXODUS project to integrate these "standalone" applications into a unified, coordinated intelligent operations support system. EXODUS will be constructed using SOCIAL, a tool for developing distributed (intelligent) systems. This paper describes EXODUS, SOCIAL, and initial prototyping efforts using SOCIAL to integrate and coordinate selected EXODUS applications.

Section 1 Introduction

Over the past decade, NASA Kennedy Space Center (KSC) has developed knowledge-based systems to increase automation of operations support tasks for the Space Shuttle fleet. Major applications include: monitoring, fault isolation and management, and control of vehicle and ground support systems; operations support of the Shuttle Launch Processing System (LPS); and planning and scheduling of Shuttle and payload processing activities.

Initial prototypes have been tested successfully (off–line) in support of several Shuttle missions. KSC is currently extending and refining these systems for formal field testing and validation. The final deployment phase of development will integrate the knowledge–based applications, both with one another and with existing Shuttle operations support systems.

Integration will require solutions to many challenging problems. KSC's knowledge–based applications were developed independently of one another, using different representation schemes, reasoning and control models, software and hardware platforms. Knowledge and data bases are application–specific, as are external interfaces to users, LPS software, and LPS data channels. In addition, KSC's knowledge–based applications lack capabilities for modeling their peer systems and for communicating with one another across heterogeneous host platforms. This precludes working together cooperatively, for example, by sharing information and by coordinating complementary activities, to solve problems that the systems are incapable of resolving individually.

KSC has recently initiated the EXODUS project (Expert Systems for Operations Distributed Users) to investigate and address these difficult issues. A high–level integration architecture has been designed. The design incorporates a hierarchical distributed control model to coordinate cooperative efforts among KSC's intelligent operations support applications. In order to refine, test, and implement this design, KSC is funding Symbiotics, Inc. to develop SOCIAL, a generalized tool for integrating and coordinating distributed systems comprised of heterogeneous intelligent and conventional elements. Symbiotics is also developing proof–of–concept prototypes to validate SOCIAL and the proposed EXODUS architecture.

The remaining sections of this paper describe, in order: the EXODUS problem domain and system design; the SOCIAL development tool; and the demonstration prototypes that integrate and coordinate selected knowledge–based applications at KSC.

Section 2 EXODUS

2.1 Space Shuttle Ground Operations

Processing, testing, and launching of Shuttle vehicles takes place at facilities dispersed across the KSC complex, often using complex Ground Support Equipment. For example, Orbiters are mated to external tanks and solid rocket engines using cranes at the Vehicle Assembly Building. Propellant storage and loading systems are used to fuel Shuttle vehicles mounted on Mobile Launch Platforms at Launch Pads.

The Launch Processing System (LPS) supports all Shuttle preparation and test activities from arrival at KSC through to launch. The LPS provides the sole direct real–time interface between Shuttle engineers, Orbiter vehicles and payloads, and associated Ground Support Equipment [He87]. Four independent physical copies, called *Firing Rooms*, can support simultaneous processing of multiple Shuttle vehicles, LPS software development, and launch team training.

A Firing Room is an integrated network of computers, software, displays, controls, switches, data links and hardware interface devices (cf. Figure 1). The computers in a Firing Room are organized in a star network. The star's locus, called the Common Data Buffer, collects data, transfers data to LPS peripheral storage subsystems, and mediates

computer–to–computer communications, which are concurrent and asynchronous. During peak (launch) conditions, a Firing Room handles thousands of commands and measurements per minute.

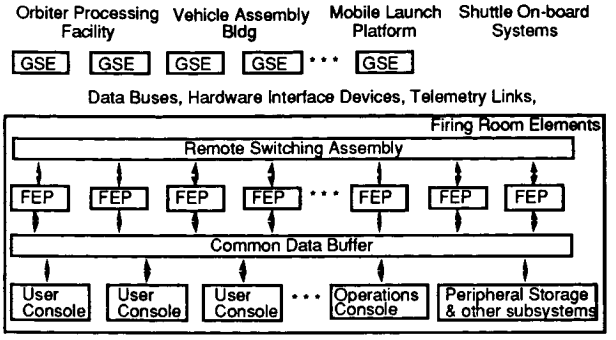

Figure .1: Architecture of an LPS Firing Room

Firing Room computers are configured to perform independent LPS functions through application software loads. Shuttle engineers use computers configured as Consoles to remotely monitor and control specific vehicle and Ground Support systems. Each such application Console communicates with an associated Front–End Processor computer that issues commands, polls sensors, and preprocesses sensor measurement data to detect significant changes and exceptional values. These computers are connected to data busses and telemetry channels that interface with Shuttles and Ground Support Equipment through switching assemblies in each Firing Room.

2.2 EXODUS Applications

EXODUS will integrate and coordinate knowledge–based applications that span KSC's major processing functions - Shuttle and LPS operations and planning and scheduling of such operations. Tasks in all three areas are labor– and expert–intensive. KSC's intelligent systems program will: increase automation of operations support tasks, alleviating labor requirements and costs; improve safety by standardizing (expert) task performance and increasing accessibility of data on problems and problem solutions; and preserve expertise that would otherwise be lost when veteran NASA engineers change jobs or retire. This section summarizes the primary KSC applications in the EXODUS framework.

The LPS Operations team ensures that the four Firing Rooms are available continuously, in appropriate error–free configurations to support Shuttle engineering test requirements such as Launch Countdown or Orbiter Power-up sequences. OPERA (for Operations Analyst) consists of an integrated collection of expert systems that automates some of these critical support functions [Ad89b].

OPERA's primary expert system monitors a Firing Room for anomalies and assists LPS Operations users in isolating and managing faults by recommending troubleshooting, recovery and/or workaround procedures. OPERA taps into and interprets a data stream comprised of error messages triggered by the LPS Operating System. Messages signal anomalous events such as improper register values or expiring process timers. OPERA also incorporates two secondary expert systems, which interface with and maintain data and knowledge bases that track open and recurring problems across all four Firing Rooms. They assist the primary expert by retrieving fault reports that provide relevant precedents to current problem symptoms.

The LPS Operations team replaces problem Firing Room computers with standby spares to restore on–line functionality to Shuttle engineering end–users. Suspect or faulty computers are then diagnosed and repaired off–line by an LPS Maintenance organization, which is developing a supporting Remote Monitoring and Maintenance Subsystem (RMMS). RMMS consists of custom hardware implants that capture memory dumps from failing Firing Room computers, and a tap to the Common Data Buffer for retrieving and storing dump data files. An associated Memory Dump Analyst provides an object–oriented interface for inspecting memory dumps and a shallow–knowledge expert system that automatically diagnoses a subset of computer faults.

KSC has developed a model–based tool called KATE (or Knowledge Based Autonomous Test Engineer) for building intelligent systems to automate monitoring, diagnosis, and control tasks for Shuttle Ground Support Equipment [Fu90]. These systems are comprised of electromechanical components including relays, pumps, blowers, ducts, heaters, and embedded sensors. KATE extends and generalizes on LES, an early model–based diagnostic system that supports the the Liquid Oxygen fuel loading system [Sc87].

KATE applications monitor Firing Room Console data while simultaneously running a behavioral model simulation for their target Ground Support Equipment system. Discrepancies between actual data and values predicted by the model trigger the model–based diagnostic module. Control capabilities can be used to test diagnostic hypotheses (via sensor requests) and to issue corrective commands. A KATE–based application called LOX (an extended reimplemented version of LES) is currently being validated in field tests. Another KATE system (ECS) has been developed to help maintain environmental controls for the Shuttle cargo bay when the vehicle is at a Launch Pad.

EXODUS will also integrate knowledge–based tools for planning and scheduling resources and activities for payload integration and Shuttle processing [Mu88,Zw89]. Further expert systems are being designed to assist LPS Operations in configuring Firing Room switching assemblies and to automate Shuttle engineering activities at application Console stations.

2.3 EXODUS Architecture

LPS Firing Room (ModComp-II) computers were built in the early 1970s. Their limited memory capacity is largely occupied by LPS Operating System and Shuttle user application software. Accordingly, KSC's knowledge–based systems have been implemented on other platforms, including Sun Workstations, Texas Instruments Explorer Lisp Machines, and PCs.

The proposed EXODUS architecture (cf. Figure 2) will use an Ethernet local area network for physically connecting intelligent application hosts. Intelligent systems will access LPS Firing Room data via an interface between the Common Data Buffer and a data concentrator. This interface currently extracts memory dump data for RMMS and Operating System error messages for OPERA. Extensions to support data and

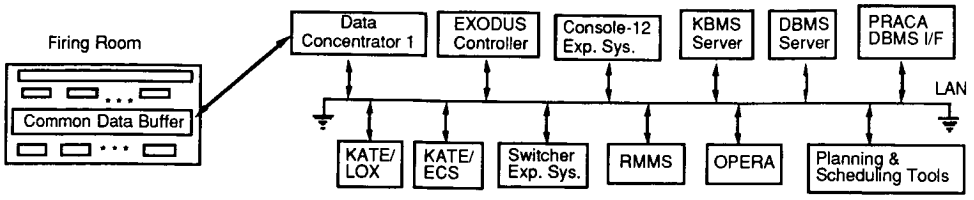

Figure .2: EXODUS Architecture

control interfaces for KATE applications are being designed. A centralized interface design is necessary for two reasons: (a) the limited number of free ports into Common Data Buffers; and (b) the major testing effort is required to validate and verify new LPS interfaces with respect to NASA's stringent safety requirements.

The proposed integration design for EXODUS adopts a server–based architecture: critical data and knowledge bases in EXODUS applications will be redistributed to server nodes comprised of dedicated data and knowledge base management systems running on high performance, large memory capacity hardware platforms. This design approach promotes sharing of symbolic models of common utility across applications: Shuttle and LPS system structures, behaviors, and bodies of operational expertise. Maintenance, access control, and commonality of interfaces will also be facilitated.

Redistributing large data and knowledge bases to server platforms will also reduce memory and performance burdens from EXODUS applications on their hosts. This will become critical since plans call for porting EXODUS applications over to the new Console computers being procured for a modernization of Firing Rooms in the mid–1990s.

The critical requirements for the proposed EXODUS integration architecture are: (a) non–intrusive communication capabilities for moving data and commands among heterogeneous applications and information resources; and (b) intelligent distributed control models to coordinate the activities of EXODUS applications. The following sections describe development efforts for these enabling technologies.

Section 3 The SOCIAL Development Tool

Obstacles to integrating "standalone" intelligent systems are not unique to KSC or to operations support. Analogous difficulties arise in other domains including: battle management; decision support; manufacturing process control; air traffic control; concurrent engineering environments; power generation plants; and power transmission and communication networks.

These domains encompass multiple problems of varying complexity, whose solutions may be independent or only weakly dependent upon one another. Different problem–solving architectures are appropriate for disparate tasks. Complex computer systems already exist for storing data and executing conventional programs that automate routine activities (e.g., for sensor and equipment control, instrumentation, event trapping, and bounded scheduling tasks). Software and hardware platforms are typically heterogeneous across intelligent and conventional applications. Finally, a priori design of comprehensive integration strategies was generally infeasible in the technology development or transfer environments where intelligent systems currently being deployed were initiated.

SOCIAL is a generalized tool that is being built for developing distributed systems and for integrating existing systems "after the fact" [Ad89a,Ad90]. SOCIAL will provide the following broad functional capabilities and attributes:

- a high–level, modular distributed communications capability for passing information between applications based on heterogeneous languages, platforms, networks, and network protocols. This subsystem is already available as a standalone commercial product called *MetaCourier*;
- minimally intrusive data and control interfaces to new and existing systems, both conventional and intelligent, including data feeds and applications developed using commercial AI shells and relational database management systems (RDBMSs);
- portability across heterogeneous software and hardware platforms;
- predefined intelligent control models to coordinate cooperative problem–solving activities of distributed (knowledge based) applications with heterogeneous internal control and communication architectures;
- tools for customizing and extending existing control models and interfaces.

SOCIAL's architecture is based on a layered library of object–oriented building blocks. The highest level objects are called *Agents*. Distributed systems are constructed by instantiating suitable Agent types, embedding application elements in these instances, and connecting the resulting Agents together. Agent instances provide generic distributed services to their embedded application elements. These services, implemented via lower–level object–oriented building blocks, include distributed communication, data and knowledge access, and control (e.g., process coordination, concurrency and reliability management).

Application elements access the distributed services of their embedding Agents through a high–level *Message–based* interface. For example, an application communicates with another via messages of the form (Tell :agent X :system Y message-contents). For each application Agent, the developer must define the expected form of incoming messages (i.e. an argument list), along with three procedural methods that specify: how to parse and process messages; test predicates for determining completion (i.e., in case the Agent dispatches messages to one or more other Agents for intermediate processing); and what

results the embedded receiving application is to return. Auxiliary methods can be defined to simplify the organization of these primary Agent methods.

Message protocols determine the kind of communication behavior required for Agent interactions. The "Tell" protocol signals asynchronous behavior whereas "Tell–and–Block" indicates synchronous, "wait–and–see" behavior: an Agent that sends a Tell message can go on to perform other tasks pending returning information, whereas a Tell–and–Block message implements a function call and return control model.

All distributed control and information access behaviors are defined in terms of MetaCourier's message–based communication services, the substrate layer of the SOCIAL architecture. Distributed control is achieved through Agents autonomously invoking other Agents. For example, concurrency is accomplished by asynchronous message-passing to invoke multiple Agents more or less simultaneously. Similarly, parallelism amounts to dispatching subtasks (single or multiple instruction with multiple data) to a set of server Agents with a *broadcast* protocol of batched Tells. Non–intrusive access to, and integration of, passive data resources and existing standalone applications is accomplished through "wrapper" Agents that define suitable external command and data interfaces.

SOCIAL's message–based interfaces enforce a clean partitioning between application–specific functionality and predefined services such as distributed communications. To ensure portability, SOCIAL further isolates Agent dependencies on processing platforms, networks, and software environments (e.g. cpu, operating system, network type and host address, language compiler and editor), in separate (shared) "Host" and "Environment" objects. SOCIAL's MetaCourier subsystem uses message protocols and Host and Environment objects associated with the sending and receiving Agents to determine how to transmit messages across heterogeneous hardware and software platforms transparently. By separating and concealing the mechanical complexities of distributed processing, SOCIAL frees developers to concentrate on the architecture and behavior of their distributed applications. The first version of SOCIAL is scheduled to be completed at the end of 1990.

Section 4 EXODUS Prototypes

4.1 Distributed Data Transfer

The Data Concentrator is a critical component in the EXODUS architecture. It must concentrate, classify, and route real–time data to the intelligent subsystems responsible for monitoring Ground Support Equipment and Firing Rooms and isolating faults. A proof–of–concept simulation of these data transfer functions was constructed using SOCIAL. Figure 3 depicts the Firing Room data sources, EXODUS knowledge–based systems, and and hardware and software platforms for those systems. Network connections consist of Ethernet media and TCP/IP protocols.

The client/server Remote Procedural Call (RPC) model is the de facto communications standard today. This model is inherently synchronous, asymmetric, and pairwise: active clients request and block for services from reactive servers and a given client can only interact with a single type of server. Synchronous processing is unsuitable for the high volume data transfers required by EXODUS. The client-server model also forces the (active) data concentrator to be modeled as a router that sorts and feeds data to a set of client processes that use "requests" to transmit the data to (passive) EXODUS "server" applications. In contrast, SOCIAL's MetaCourier layer provides an asynchronous, symmetric, and peer–to–peer model. A single Agent can act as a client or a server or operate in both roles, and a "client" Agent can interact with multiple "server" Agents. In addition, behaviors can be inherited and/or specialized across Agent types.

The EXODUS simulation defines a single Data Concentrator Agent and a class of Data Injector Agents that are co–resident with the various intelligent Operations Support applications. The Data Concentrator receives and preprocesses Firing Room data. The concentrator agent then classifies and encapsulates the resulting data in MetaCourier messages, which are dispatched directly and asynchronously to relevant Injector Agents. Injectors inherit the structure and functionality of the Injector Agent class, specialized by a single dispatch method for injecting the data to the input interface for a particular application. The OPERA Data Injector, shown below, inserts data into a First-In-First-Out input buffer for CCMS Operating System messages. The RMMS Memory Dump Analyst Injector simply notifies users that new computer memory dumps are available for inspection.

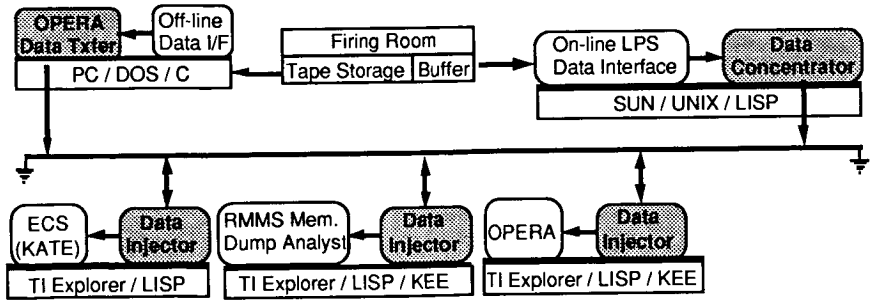

Figure .3: SOCIAL Exodus Data Transfer Simulation

```
(defagent OPERA-DATA-INJECTOR
  :sys *opera-host*   ;;;OPERA host (vble)
  :environ 'exodus
  :args ($datum)      ;;;msg structure
  :lifetime :image
  :type (data-injector)  ;;; Agent class
  :documentation
  "This Agent inserts LPS Operating System
   error messages into the FIFO queue that
   serves as the OPERA LPS Data Interface"
  :in-filter ;;; inherited method/behavior
             ;;; to process incoming msg
 (sendx :self :dispatch-datum $datum)
  :methods
  ;;;OPERA-specific injector data interface
  ((:dispatch-datum ($data)
     (unless (string= $data "")
        (eval '(kee::add.value
          'kee::opera-controller
          'kee::opera-ccms-data-interface
          ,$data))))))
```

4.2 Distributing OPERA's Expert Systems

The capability to distribute a complex intelligent application across multiple platforms is critical for realizing EXODUS's resource server architecture. Physical distribution is clearly important for performance: time–intensive processes that search rule–bases or databases should be isolated, allocating dedicated computing resources to critical functions such as real–time data monitoring. Distribution of large knowledge bases also reduces memory loading. Because EXODUS encompasses existing applications, it must also be possible to *redistribute* application elements transparently and non–intrusively.

To demonstrate these capabilities, SOCIAL was used to physically distribute the OPERA system. OPERA is a logically distributed system that integrates and coordinates multiple expert systems that were originally developed as co–residents on a single platform. A control module coordinates the activities of OPERA's expert systems and manages all external interfaces. Expert systems request services from the Controller, which routes those tasks to appropriate servers. Expert systems post and retrieve task results from a shared memory "Bulletin–Board" on the Controller. OPERA's expert systems and Controller are integrated by embedding them within instances of a generic distributed blackboard structure, which provides standardized communications protocols [Ad89c].

Physical (re)distribution of OPERA elements was accomplished as follows (cf. Figure 4). The three primary blackboard protocols were altered to redirect communications as messages to MetaCourier Agents rather than as postings to other blackboards. Second, the OPERA Controller's service request routing table was extended to indicate a MetaCourier agent and host platform for each OPERA subsystem/blackboard. Third, MetaCourier Agents were written for each blackboard. The action of those Agents is simply to execute a protocol behavior that posts a message as an entry to the relevant structure on their associated blackboard. Finally, because distributed expert systems no longer have direct access to all OPERA information, additional messages were built into the protocols to ensure that information required to perform tasks was transmitted prior to task requests.

The redistribution experiment required roughly four days and one hundred lines of code. Extending the blackboard architecture using SOCIAL was quite simple. However, difficulties arose because the expert systems that were distributed depended on several common utility functions and data structures that were scattered across multiple source files and knowledge bases. The lesson drawn from this exercise is that these dependencies should be tracked as part of a standard development discipline for distributed systems. Such specifications would greatly simplify the (re)organization of system code and the identification of knowledge structures that need to be copied remotely.

4.3 Distributed Data and Knowledge Access

A third EXODUS requirement will be tools for developing non–intrusive interfaces to standalone applications and information resources. SOCIAL is addressing this need through "wrapper" Agent Types called Receptionists, which define bidirectional interfaces for passing control (i.e., commands), and data to the embedded resource or program.

Databases and application programs are often constructed using commercial development tools. The design of Receptionists for such systems can be simplified by abstracting the application-

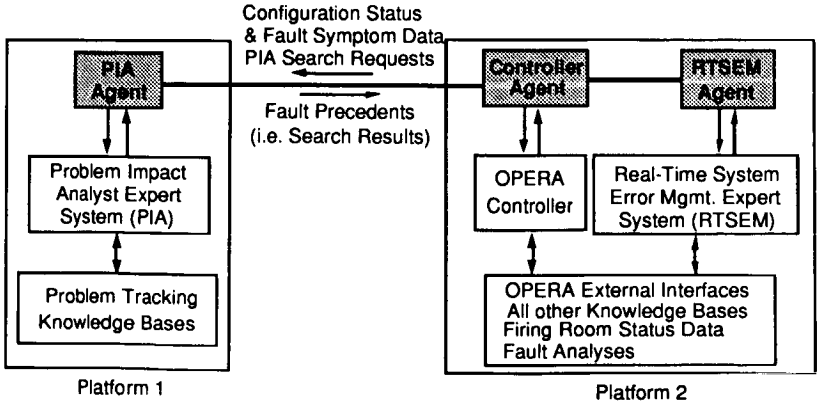

Figure .4: Using SOCIAL to Distribute OPERA

independent aspects of the control and data interface into a standardized, specialized Receptionist Agent type called a Gateway. Integrating an application element using a Gateway reduces to defining the application–specific aspects of the interface: the Gateway understands predefined query and command types which developers use to write specific queries or commands that name particular application objects and object attributes.

The basic operation of Gateways (or Receptionists) is depicted in Figure 5. An application's Agent sends a message to a Gateway Agent to access a protected resource or program. Depending on the situation, messages might contain data queries (i.e., read or write requests), or other commands to an application. Queries and commands may be expressed in a uniform, canonical language. An intelligent system might initiate queries or commands in its own development environment language through its Gateway to other SOCIAL Agents (including other Receptionists).

Gateways contain an interface library that maps canonical SOCIAL queries and commands into the language format of the relevant DBMS or shell environment, and vice versa. (Commands can be formulated in the target system's native language if desired, and will be passed through without alteration.) Gateways will also manage common exceptions (e.g., failed references or transactions), platform–specific data type conversions, and security features for restricting access to authorized Agents.

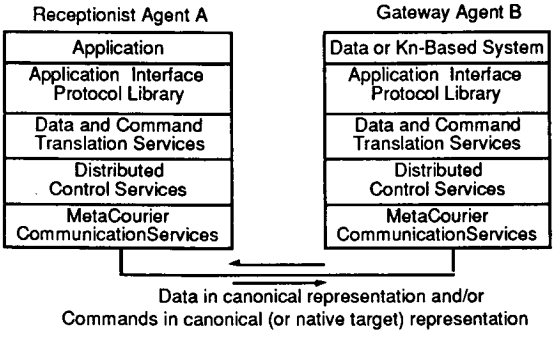

Figure .5: Integrating standalone systems using Gateways

An EXODUS simulation (cf. Figure 6) is currently being designed and implemented to demonstrate Gateway Agents for KEE, a LISP–based AI shell, CLIPS, NASA's C–based rule shell, and an Oracle relational DBMS. Briefly, OPERA will receive LPS error messages that indicate a failure in a Firing Room computer. OPERA will then request a reconfiguration action from the expert system for the Firing Room Switching Assembly. OPERA will then update its model of the Firing Room based on the Switcher expert system and formulate error report entries to the Problem–Tracking Database.

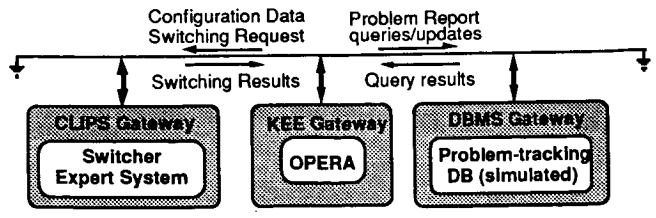

Figure .6: Distributed Data/Knowledge Access for EXODUS

4.4 Distributed Control

Aside from a robust communications substrate to provide the basic integration framework, the most important functional requirement for EXODUS is a capability to coordinate the activities of member applications. The proposed EXODUS architecture calls for a hierarchical distributed control model: a high–level Controller module will direct the intelligent applications described in Section 2 based on a global model of EXODUS subsystems, their associated KSC operations subdomains, and their relationships to one another.

SOCIAL will address this requirement through Agent types called Managers. A Manager Agent identifies all member (or subordinate) Agents by logical name and location, and also defines a distributed control model for organizing member Agents to work together cooperatively. It may also define specialized communication protocols for its members (e.g., one–to–many broadcast), and manage communication between member and outside Agents. Managers often provide a shared memory store of current problem–solving data for its members. Finally, Manager Agents may themselves be members of more complex organizations, subordinate to other Manager Agents.

The first Manager Agent type to be built for SOCIAL will be a reimplementation of OPERA's hierarchical distributed blackboard model (HDB) [Ad89c]. The HDB incorporates a routing table of member Agents describing their services and locations. The HDB also contains a centralized Bulletin–Board for expert systems to post service requests and post and retrieve request responses. The HDB control model routes all posted requests to suitable servers and orders and controls the activations of member expert system Agents. Member Agents can only communicate with one another indirectly, through the HDB Manager, using a common set of utility protocols for posting tasks to the HDB Manager's Agenda and posting results or checking for results on the HDB Manager's Bulletin–Board.

An EXODUS prototype is being planned that will utilize a Controller based on SOCIAL's HDB Manager Agent (cf. Figure 7). This Agent will coordinate KSC's intelligent systems for Shuttle and LPS Operations support to collectively solve a fault isolation problem that no single system could resolve individually. A test scenario will be defined in terms of LPS Operating System messages, Ground Support Equipment data, and Firing Room CPU memory dumps. The test scenario will simulate a Firing Room problem that may be caused by one of several possible fault candidates.

The EXODUS Controller will initialize member Agents and the Data Concentrator interface to a Firing Room. OPERA will process LPS error messages and inform the Controller of possible Firing Room anomalies. Because Firing Rooms lack adequate built–in test capabilities, OPERA can isolate fault candidates but cannot test them to produce an actual diagnosis. The EXODUS Controller will invoke the RMMS Memory Dump analyst expert system to investigate the possibility of a problem Console computer and also check KATE/LOX Agent to investigate the possibility of a failure in the Liquid Oxygen Subsystem. It will then use the hypothesis test results to reduce the set of fault candidates and display the results to Operations users.

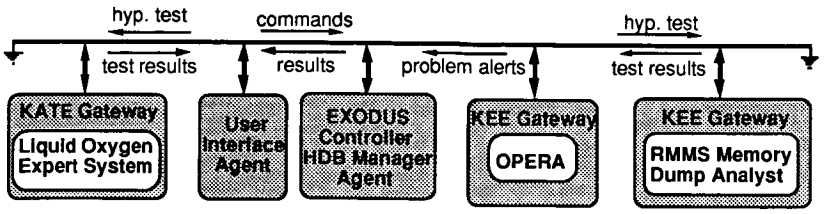

Figure .7: SOCIAL Exodus Distributed Cooperative Control

Summary

NASA Kennedy Space Center has initiated the EXODUS project to integrate and coordinate knowledge–based systems that are helping to automate Ground Operations activities in support of the Space Shuttle fleet. Individual applications were designed for "standalone" use with heterogeneous architectures, languages, and hardware platforms. Similar requirements exist for integrating conventional and knowledge–based systems in other Government and commercial domains. To minimize costly re–engineering, generalized integration tools must be developed that are non–intrusive, modular, and extensible.

KSC is using the SOCIAL development tool from Symbiotics, Inc. in the EXODUS effort. SOCIAL enforces a clear separation between application–specific functionality and standardized services for distributed communication, control, and data and knowledge access. Application elements invoke these services through high–level message–based interfaces to "wrapper" Agents, concealing the complexity and heterogeneity of the underlying distributed computing mechanisms and processing environments.

Proof–of–concept prototypes are described for validating the proposed EXODUS architecture using SOCIAL . These prototypes demonstrate SOCIAL's capability to support nonintrusive: distributed data transfer; physical distribution of a complex application comprised of previously co–resident expert systems and knowledge bases; cooperation of expert systems and data bases across multiple development tools; and hierarchical distributed coordination of standalone intelligent systems to solve difficult problems collectively.

Acknowledgments

Development of MetaCourier has been sponsored by the U.S. Army Signal Warfare Center, under Contract No. DAAB10-87-C-0053. Development of SOCIAL has been sponsored by NASA Kennedy Space Center under contract No. NAS10-11606. Astrid Heard initiated and oversees the EXODUS project at KSC. Rick Wood designed and implemented SOCIAL's distributed data and knowledge access capabilities. Pat Pinkowski and R. Bruce Hosken have provided valuable assistance in preparing EXODUS demonstrations at KSC.

Bibliography

[Ad89a] R.M. Adler, B. H. Cottman. "A Development Framework for Distributed Artificial Intelligence." *Proceedings Fifth Conference on AI Applications, Computer Society of the IEEE.* Miami, FL, March 6-10, 1989.

[Ad89b] R.M. Adler, A. Heard, and R. B. Hosken. "OPERA - An Expert Operations Analyst for A Distributed Computer Network." *Proceedings Annual AI Systems in Government Conference, Computer Society of the IEEE.* Washington, D.C., March 27-31, 1989.

[Ad89c] R.M. Adler. "A Distributed Blackboard Architecture for Integrating Loosely-Coupled Knowledge-Based Systems." *Intelligent Systems Review.* 1, 4, Summer, 1989, Association for Intelligent Systems Technology, E. Syracuse, NY.

[Ad90] R.M. Adler, B. H. Cottman. "A Development Framework for AI Based Distributed Operations Support Systems." *Proceedings Fifth Conference on AI for Space Applications.* Huntsville, AL, May 21-23, 1990.

[Fu90] S. Fulton and C. Pepe. "An Introduction to Model–Based Reasoning." *AI Expert.* 5, 1, January, 1990.

[He87] A.E. Heard. "The Launch Processing System with a Future Look to OPERA." *Acta Astronautica.* IAF-87-215, 1987.

[Mu88] A.M. Mulvehill. "A User Interface for a Knowledge–Based Planning and Scheduling System." *IEEE Transactions on Systems, Man, and Cybernetics.* SMC-18, 4, July/August 1988.

[Sc87] E.A. Scarl, J.R. Jameson, C.I. DeLaune."Diagnosis and Sensor Validation Through Knowledge of Structure and Function." *IEEE Transactions on Systems, Man, and Cybernetics.*SMC-17, 3, May/June 1987.

[Zw89] M. Zweben and M. Eskey. "Constraint Satisfaction with Delayed Evaluation." *Proceedings, 11th IJCAI.* Detroit, MI, Aug 20-25, 1989.